Requirement Elicitation

Week 3

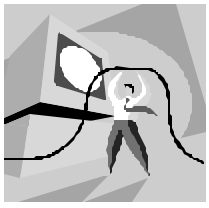
Agenda

- Problem in requirement elicitation
- Requirement process
- Types of requirements
- Fact finding techniques
- Requirement elicitation process
  - Use case model
  - Data dictionary and non-functional requirements

Requirements Elicitation

Maria in HR

"Hello, Phil? We're having a problem with the employee system you programmed for us. An employee just changed her name to Sparkle Starlight, and we can't get the system to accept the name change. Can you help?"



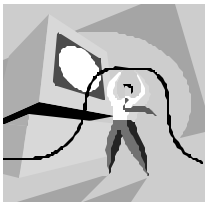
Phil in Systems

"She married some guy named Starlight?"

Requirements Elicitation (cont)

Maria in HR

"No, she didn't get married, just changed her name. That's the problem. It looks like we can change a name only if someone's marital status changes."



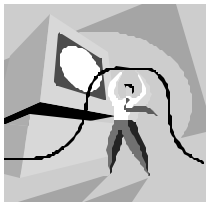
Phil in Systems

"Well, yeah, I never thought someone might just change her name. I don't remember you telling me about this possibility when we talked about the system. That's why you can get to the Change Name dialog box only from the Change Marital Status dialog box."

Requirements Elicitation (cont)

"I assumed you knew that people could legally change their name anytime they like. We have to straighten this out by Friday, or Sparkle won't be able to cash her paycheck. Can you fix the bug by then?"

Maria in HR



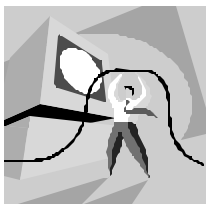
"It's not a bug! I never knew you needed this capability. I'm busy now. I can probably fix it by the end of the month. Next time, tell me these things earlier, and please write them down."

Phil in Systems

Requirements Elicitation (cont)

"What am I going to tell Sparkle? She's really going to be ticked if she can't cash her check"

Maria in HR



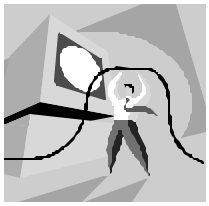
"Hey Maria, it's not my fault. If you'd told me in the first place that you had to be able to change someone's name at any time, this wouldn't have happened. You can't blame me for not reading your mind."

Phil in Systems

Requirements Elicitation (cont)

Maria in HR

"Yeah, well, this is the kind of thing that makes me hate computer systems. Call me as soon as it's fixed, will you?"



Phil in Systems

"#$%^&*@..."

What's wrong here

- Informal information gathering
- Implied functionality
- Erroneous or un-communicated assumptions
- Inadequately defined requirements
- A casual change process

Requirement

- Requirement
  - A feature that the system must have or a constraint that it must satisfy to be accepted by the client
- Two questions need to be answered
  - How can we identify the purpose of a system?
  - Crucial is the definition of the system boundary: What is inside, what is outside the system?
- These two questions are answered in the requirements process
- The requirements process consists of two activities:
  - Requirements Elicitation:
    - Definition of the system in terms understood by the customer ("Problem Description")
  - Requirements Analysis:
    - Technical specification of the system in terms understood by the developer ("Problem Specification")

Products of requirements process

- Requirement elicitation results in the specification of the system that the client understands
- Analysis results in an analysis model that the developers can unambiguously interpret

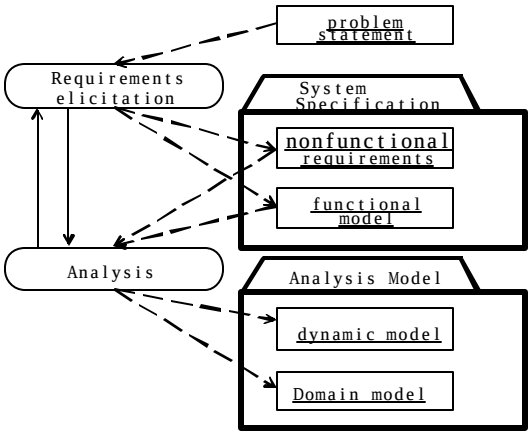


Figure 3.1 Products of requirement process

System Specification vs. Analysis Model

- Both models focus on the requirements from the user's view of the system.
- ***System specification*** uses natural language (derived from the *problem statement*)
- The ***analysis model*** uses formal or semi-formal notation (for example, a graphical language like UML)
- The starting point is the problem statement

Activities in requirement elicitation

- Identifying actors
- Identifying scenarios
- Identifying user cases
- Refining use cases
- Identifying relationships among user cases (Use case diagram)
- Identifying non-functional requirements

Types of Requirements

- FURPS+
  - **F**unctional requirements
  - Nonfunctional requirements
    - **U**sability
    - **R**eliability
    - **P**erformance
    - **S**upportability
    - **+** - ancillary and sub-factors (constraints or pseudo requirements)
      - Implementation
      - Interface
      - Operations
      - Packaging
      - Legal

Functional Requirements

- What are functional requirements
  - Describe the interactions between the system and its environment independent of its implementation.
  - Features, capabilities …
- Functional requirements are explored and recorded in use cases model
- Excerpt from functional requirements of `SatWatch`

> `SatWatch` is a wrist watch that displays the time based on its current location. `SatWatch` uses GPS satellites (Global Positioning System) to determine its location and internal data structures to convert this location into a time zone.
>
> …
>
> When political boundaries change, the watch owner may upgrade the software of the watch using the WebifyWatch device (provided with the watch) and a personal computer connected to the Internet.

Table 3.1 functional requirements of `SatWatch`

Non functional requirements

- Aspects of the system that are not directly related to the functional behavior of the system
- Quality requirements
  - **U**sability
    - The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.
  - **R**eliability
    - The ability of a system or component to perform its required functions under stated conditions for a specified period of time
  - **P**erformance
    - Quantifiable attributes of the system
    - Response time, throughput, availability, accuracy
  - **S**upportability
    - The ease of change to he system after deployment
    - Adaptability, maintainability..

Nonfunctional requirements (cont)

- Constraints
  - Implementation requirements
    - The use of specific tools, programming languages, or hardware platforms
  - Interface requirements
    - Constraints imposed by external systems, including legacy systems and interchange formats
  - Operations requirements
    - Constraints on the administration and management of the system in the operational setting
  - Packaging requirements
    - Constraints on the actual delivery of the system
  - Legal requirements
    - Concerned with licensing, regulation, and certification issues

Nonfunctional requirements for SatWatch

- Quality requirements for SatWatch
  - Any user who knows how to read a digital watch and understands international time zone abbreviations should be able to use SatWatch without the user manual. [Usability requirement]
  - As the SatWatch has no buttons, no software faults requiring the resetting of the watch should occur. [Reliability requirement]
  - SatWatch should display the correct time zone within 5 minutes of the end of a GPS blackout period. [Performance requirement]
  - SatWatch should display time correctly in all 24 time zones. [Performance requirement]
  - SatWatch should accept upgrades to its onboard via the Webify Watch serial interface. [Supportability requirement]

Nonfunctional requirements for SatWatch (cont)

- Constraints for SatWatch
  - All related software associated with SatWatch, including the onboard software, will be written using Java, to comply with current company policy [Implementation requirement]
  - SatWatch complies with the physical, electrical and software interface defined by WebifyWatch API 2.0 [Interface requirement]

Types of Requirements Elicitation

- Greenfield Engineering
  - Development starts from scratch, no prior system exists, the requirements are extracted from the end users and the client
  - Triggered by user needs
  - **Example:** Develop a game from scratch: Asteroids
- Re-engineering
  - Re-design and/or re-implementation of an existing system using newer technology
  - Triggered by technology enabler
  - **Example:** Reengineering an existing game
- Interface Engineering
  - Provide the services of an existing system in a new environment
  - Triggered by technology enabler or new market needs
  - **Example:** Interface to an existing game (Bumpers)

Discussion Question I

- Specify which of these statements are functional requirements and which are nonfunctional requirements
  1. The `TicketDistributor` must enable a traveler to buy weekly passes.
  2. The `TicketDistributor` must be written in Java
  3. The `TicketDistributor` must be easy to use
  4. The `TicketDistributor` must always be available
  5. The `TicketDistributor` must provide a phone number to call when it fails.

Fact finding techniques

- Background reading
  - Company reports, organization charts, policy manuals, job descriptions, reports and documentation of existing systems
  - Advantages and disadvantages
    - Helps the analyst to get an understanding of the organization before meeting the people who work there.
    - Allows the analyst to prepare for other types of fact findings
    - May provide formally defined information requirements for the current system
    - Do not match up to reality
  - Appropriate situation
    - When the analyst is not familiar with the organization being investigated
    - Useful in the initial stages of investigation

Fact finding techniques (cont)

- Interviewing
  - Advantages and disadvantages
    - Allows the analyst to be responsive and adapt to what the user says
    - Can probe in greater depth about the person's work than be achieved with other methods
    - Time consuming and the most costly form of fact gathering
    - Can be subject to bias if the interviewers has a closed mind about the problem
    - Different interviewees might provide conflicting information
  - Appropriate situations
    - Appropriate in most projects

Fact finding techniques (cont)

- Observation
  - Advantages and disadvantages
    - Provide first hand experience of the way that the current system operates
    - Observation can be used to verify information from other sources or to look for exceptions to the standard procedure
    - Baseline data about the performance of the existing system and of users can be collected
    - People may behave differently under observation
    - Required a trained and skilled observer for it to be most effective
  - Appropriate situations
    - Essential for gathering quantitative data about people's jobs.

Fact finding techniques (cont)

- Document sampling
  - Collect copies of blank and completed documents during the course of interviews and observation sessions
  - Carry out a statistical analysis of documents in order to find out about patterns of data
  - Advantages and disadvantages
    - Can be used to gather quantitative data
    - Can be used to find out about error rates in paper documents
    - May not be able to reflect future situation
  - Appropriate situations
    - First type always appropriate
    - Statistical approach is appropriate in situations where large volumes of data are being processed

Fact finding techniques (Questionnaires)

- Questionnaires
  - Advantages and disadvantages
    - An economical way of gathering data from a large number of people
    - Result from a well-designed questionnaire can be processed by computer
    - Good questionnaires are difficult to construct
    - Not able to follow up or probe more deeply
  - Appropriate situations
    - Views or knowledge of a large number of people need to be obtained
    - Appropriate for information systems that will be used by the general public.

Use case model

- Requirement elicitation is a very challenging activity
- Requires collaboration of people with different backgrounds
  - Users with application domain knowledge
  - Developer with solution domain knowledge (design knowledge, implementation knowledge)
- Bridging the gap between user and developer:
  - ***Scenarios:*** Example of the use of the system in terms of a series of interactions with between the user and the system
  - ***Use cases:*** Abstraction that describes a class of scenarios

Identifying scenarios

- Scenarios
  - A scenario is a specific sequence of actions and interactions between actors and the system under discussion
- Scenarios can have many different uses during the software lifecycle
  - ***Requirements Elicitation***: As-is scenario, visionary scenario
  - ***Client Acceptance Test:*** Evaluation scenario
  - ***System Deployment:*** Training scenario

Types of Scenarios

- As-is scenario:
  - Used in describing a current situation. Usually used in re-engineering projects. The user describes the system.
- Visionary scenario:
  - Used to describe a future system. Usually used in greenfield engineering and reengineering projects.
  - Can often not be done by the user or developer alone
- Evaluation scenario:
  - User tasks against which the system is to be evaluated.
- Training scenario:
  - Step by step instructions that guide a novice user through a system

How do we find scenarios?

- Ask yourself or the client the following questions:
  - What are the primary tasks that the system needs to perform?
  - What data will the actor create, store, change, remove or add in the system?
  - What external changes does the system need to know about?
  - What changes or events will the actor of the system need to be informed about?

Identifying use cases

- A **use case** is a collection of related success and failure scenarios that describe actors using a system to support a goal
- Sample use case and associated scenarios

Handle return
Main success scenario:
Alternate Scenarios:
If the credit back transaction is rejected…
If the item identifier is not found in the system…

Table 3.2 sample use case for a POS system

How to specify a use case

- Name of Use Case
- Primary actor(s)
  - Description of Actors involved in use case
    - Actors represent external entities that interact with the system
    - Actor can be human or an external system
- Stakeholders and interests List
- Preconditions
  - State what must always be true before starting a scenario in the use case
- Postconditions
  - State what must be true on successful completion of the use case
- Flow of Events (Basic flow and alternative flows)
  - Free form, informal natural language
- Special Requirements
  - Nonfunctional Requirements, Constraints
- Technology and Data Variations List
  - Technical variations in *how* something must be done

A sample use case

- Preface elements

Process Sale
Primary Actor: Cashier
Stakeholders and interests: -- sales person: wants sales commissions updated -- Government tax agencies: want to collect tax from every sale …
Preconditions: Cashier is identified and authenticated
Postconditions : Sale is saved. Tax is correctly calculated. Accounting and inventory are updated. Commissions recorded…

Table 3.3 sample use case description

A sample use case (cont)

Main Success Scenario (Basic Flows):

1.customer arrives

2. Cashier starts a new sale

3. cashier enters item identifier

….

Extensions (Alternative Flows):

*a At any time, system fails

1. Cashier restarts system
2. System reconstruct prior state

3a. Invalid identifier: System signals error and rejects entry

3-6a. Customer asks cashier to remove an item from the purchase:

1. Cashier enters item identifier for removal from sale
2. System display updated running total

Special requirements:

-- Credit authorization response within 30 seconds 90% of the time

-- Touch screen UI on a large flat panel monitor

Technology and Data Variations List:

-- 3. item identifier entered by bar code laser scanner

Goals and scope of a use case

- What is a valid use case?
  - The EBP use case guideline
    - For requirement analysis for a computer application, focus on use cases at the level of elementary business processes (EBPs)
    - EBP
      - A task performed by one person in one place at one time, in response to a business event, which adds measurable business value and leaves the data in a consistent state. E.g. Approve credit or Price order
  - A common use case mistake is defining many use cases at too low a level; that is, as a single step, subfunction, or subtask within an EBP

Use cases and goal

- An EBP-level use case is called a **user goal** - level use case
  - Find the user goal
  - Define a use case for each
- Subfunction goals and use cases
  - Sub goals that support a user goal, eg: “identify myself and be validated”
  - Use cases should only occasionally be written for these subfunction goals

Finding primary actors, scenarios and use cases

- Choose the system boundary
  - Is it just a software application, the hardware and application as a unit, that plus a person using it, or an entire organization
- Identify the primary actors and goals
- Define use cases that satisfy user goals; name them according to their goals

Identify primary actors

- Questions for identifying actors
  - Which user groups are supported by the system to perform their work?
  - Which user groups execute the system's main functions?
  - Which user groups perform secondary functions such as maintenance and administration?
  - With what external hardware or software system will the system interact?
    - See textbook for a more detailed list of questions

Primary actor and system boundary

- Primary actor and user goals depend on system boundary

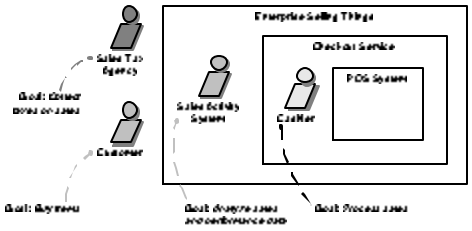


Figure 3.2 Primary actors, user goals and system boundary

Types of actors

- Primary actor
  - Has user goal fulfilled through using services of the system under discussion
  - To find user goals and drive the use case
  - Example: cashier
- Supporting actor
  - Provide a service to the system under discussion
  - To clarify external interfaces and protocols
  - Example: automated payment authorization service
- Offstage actor
  - Has an interest in the behavior of the use case, but is not primary or supporting
  - Ensure all necessary interests are identified
  - Example: government tax agency

Use case diagram

- Use case diagram helps to identify the relationship among actors and use cases
- Use case diagram and use case relationships are secondary in use case work. Use case are text documents. Doing use case work means to write text

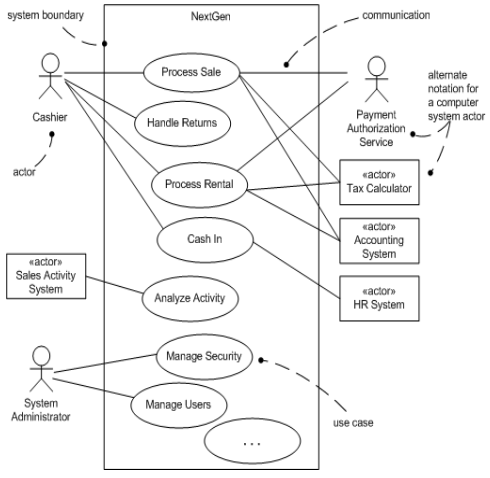


Figure 3.3 use case diagram for a POS system

Relating use cases

- Important types of use case relationships: Include, Extends
- Include
  - A use case uses another use case (“functional decomposition”)
- Extend
  - A use case extends another use case

<<Include>> functional decomposition

- Problem:
  - A function in the original problem statement is too complex to be solvable immediately
- Solution:
  - Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into smaller use cases

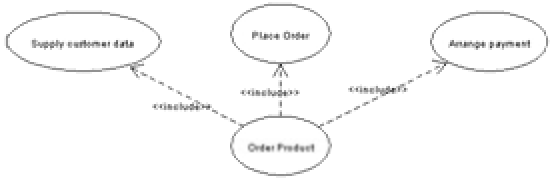


Figure 3.4 include relationship for functional decomposition

<<include>> reuse of common functionality

- Problem
  - Some partial behavior is common across several use cases
- Solution
  - The *include association* from a use case A to a use case B indicates that an instance of the use case A performs all the behavior described in the use case B

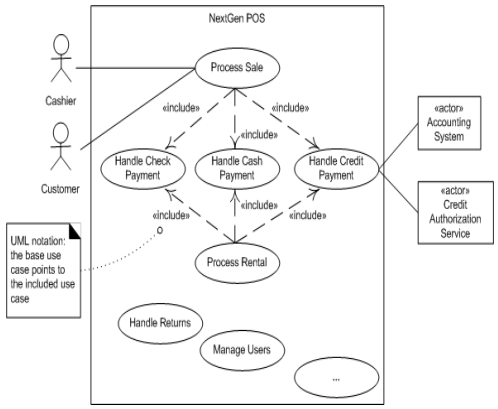


Figure 3.5 <<include>> relationship for reuse common functionality

Express include relationship in use case text

Process Sale
…
Main Success Scenario:
1.Customer arrives
…
7. Customer pays and system handles payment
Extensions:
7b. Paying by credit: *include handle credit payment*
7c. Paying by check: *include handle check payment*

Table 3.4 base use case description

Handle Credit payment
…
Level: subfunction
Main Success Scenario:
1. Customer enters their credit account information
2. System sends payment authorization request to an external payment authorization service system and request payment approval
3…
Extensions:
2a. System detects failure to collaborate with external system
…

Table 3.5 "included" use case description

<<extend>> existing use cases

- Problem:
  - The functionality in the original problem statement needs to be extended.
- Solution:
  - An *extend association* from a use case A to a use case B indicates that use case B is an extension of use case A.

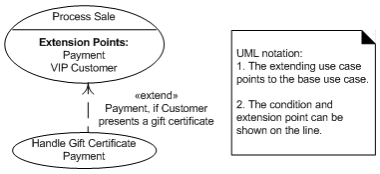


Figure 3.6 Extend relationship

Express extend relationship in use case text

Process Sale
Extension Point: *VIP customer*, step 1. *payment*, step 7.
Main Success Scenario:
1. Customer arrives at a POS checkout with goods and/or services to purchase
7. Customer pays and system handles payments

Table 3.6 base use case description

Handle gift certificate payment
…
Trigger: Customer wants to pay with gift certificate
Extension point: Payment in Process Sale
Level: subfunction
Main Success Scenario:
1. Customer gives gift certificate to cashier
2. Cashier enters gift certificate ID

Table 3.7 extending use case description

Extend vs. *include* relationships

- Direction of the relationship
  - For include relationship, the event triggering the target use case is described in the flow of event of the source use case
  - For extend relationships, the event triggering the source (the extending) use case is described in the source use case. The base use case have no knowledge of the extending use case
- The purpose of adding *include* and *extend* relationships is to reduce or remove redundancies from the use case model

Heuristics for extend and include relationship

- Use extend relationships for exceptional, optional or seldom-occurring behavior
- Use include relationship for behavior that is shared across two or more use cases
- Do not over structure the use case model. A few longer use case (two pages long) are easier to understand and review than many short ones (eg. 10 lines long)
- Always use the *include* relationship between use cases, rather than a mixture of *include* and *extend*.

Identifying initial analysis objects

- How to bridge the terminology gap between developers and users
- Identify the participating objects for each use case and collect them into a Glossary (data dictionary)

Heuristics for identifying initial analysis objects

- Terms that developers or uses must clarify to understand the use case
- Recurring nouns in the use cases
- Real-world entities that the system must track
- Real-world processes that the system must track
- Data sources or sinks
- Always use application domain terms

Term	Definition and Information	Alias
Item	A product or service for sale	
Payment authorization	Validation by an external payment authorization service that they will make or guarantee the payment to the seller	
UPC	12 digit code that identifies a product. Usually symbolized with a bar code placed on products. ..	Universal Product Code

Table 3.8 data dictionary sample

Identifying nonfunctional requirements

- Nonfunctional requirements typically includes conflicting requirements
- Few systematic methods for eliciting nonfunctional requirements.
- In practice, analyst use a taxonomy of nonfunctional requirements to generate check lists of questions to help the client and the developer on the nonfunctional aspects of the system.
- Example questions for eliciting nonfunctional requirements (see handout)

Documenting Requirement Elicitation

- The Results of the requirements elicitation and the analysis activities are documented in the **Requirement Analysis Document** (RAD)
- Introduction
- Current System
- Proposed system
  - Overview, functional requirements, nonfunctional requirements, system models.
- Glossary

Discussion questions II

- Getting information from interview transcript
- Please refer to the class handout